university of groningen / faculty of science and engineering

A tableau prover for GL provability logic

Bachelor's Project Thesis

Tim van Loo, s1987364, t.van.loo.1@student.rug.nl,
Supervisor: prof dr L.C. Verbrugge

Abstract: A simple tableau prover using sound and complete tableau-rules for the provability logic GL was implemented in C using a priority queue for the application of rules. Two types of substitutions were tested, the first substitutes each operator but the box, negation, and disjunction, and the second converts the input to negation normal form. For some input sentences, the tableau and time become shorter after substitution, where for some others it increases exponentially. Although the program typically runs in the order of magnitude microseconds for simple tableaux, the lack of benchmark sets for GL as well as some hardware limitations make the results ungeneralisable.

1 Introduction

In this research, a tableau prover will be made for provability logic, which will be used to study how various substitutions in the input influence the runtime of the program. Provability logic is a form of modal logic, but where in most modal logics the meaning of $\Box A$ is interpreted as "A is necessary", in provability logic the true meaning of $\Box A$ is closer to the interpretation "A is provable". When the word provable is used in this context, what usually is meant is provable in a formal theory like Peano Arithmetic. This is also where most applications of provability logic are; it can be used to check the soundness of proofs in Peano Arithmetic. When something is provable in provability logic, we know it must also be in Peano Arithmetic.

1.1 Peano Arithmetic and Provability Logic GL

Peano Arithmetic (PA) is a type of mathematical logic that reasons with only natural numbers (positive integers) and logical operations on these natural numbers. Even though PA itself is an abstract concept, various proofs in different scientific fields can be constructed in it. The notion of provability is also reflected within PA itself, in the form of the provability function $Bew(\ulcorner A \urcorner)$, which stands for "A is provable in PA" ("Bew" is short for "beweisbar", the German word for "provable"). This Bew-function is closely related to the $\Box$-operator in provability logic, as the fundamental idea behind provability logic is that the $\Box$-operator of modal logic could be made to stand for provability in a formal system. The type of provability logic that will be used in this research is called GL (after Gödel and Löb), and its $\Box$-operator has indeed been shown to be equivalent to the provability function Bew in Peano Arithmetic (Solovay, 1976).

Formally, GL is an extension of the basic modal logic K with the Löb axiom $\Box(\Box A \rightarrow A) \rightarrow \Box A$. The addition of this axiom is what changes the meaning of the $\Box A$ operator, but, like in K, the $\Diamond A$ is still defined in terms of the $\Box$-operator ($\Diamond A := \neg\Box\neg A$). Using this definition, $\Diamond A$ can be interpreted as "not A is not provable", or in other words "A is consistent".

1.2 Tableaux for GL

One of the easiest (and most comprehensible) ways to show if a formula is provable in GL, would be by constructing a semantic tableau using Kripke semantics, because this makes it easy to read out a counter-model if one exists. In this kind of tableau, each branch represents a Kripke model. A Kripke model consists of a set of worlds, a set of accessibility relations between those worlds, and a set of truth values for propositions in the worlds. In order to prove a sentence A in GL, a semantic tableau can be constructed for $\neg A$. If each branch is closed af-

ter application of all rules (i.e. each branch contains a contradiction), then A has been shown provable in GL (proof by contradiction). However, if one or more branches on the tableau are still open, each of those branches can serve as a counter-model for A, since they represent models where $\neg A$ is true (so A is not valid). (Boolos, 1995)

For the construction of a world-labelled tableau, sound and complete rules for GL will be needed. Since GL extends the basic normal logic K, all K rules still apply. For the basic operators, this means their rules can be used unchanged. For the modal operators however, the rules will have to change slightly, since they are no longer complete after addition of the GL axiom to K.

1.3 Research Question

In a tableau prover for provability logic that runs in PSPACE, does modifying the input to a logically equivalent formula using less operators before constructing the tableau influence run time?

When describing modal calculus rules for the construction of semantic tableaux, a lot of literature (e.g. Rautenberg (1983), Goré (1999)) seems to only include $\Box$. Of course, since in any formula all appearances of $\Box$ can be substituted for $\neg\Diamond\neg$, explicitly describing the rule for $\Diamond$ is not needed and using it can be avoided altogether by substituting all appearances of $\Diamond$ with $\neg\Box\neg$.

Similarly, when implementing a tableau prover, one might only allow as few as three operators in the input (e.g. $\Box, \vee, \neg$), in an attempt to improve the run time of the program by limiting the amount of possible rules. The downside of such an approach is that it first requires the user to substitute all appearances of other operators in the input, which may make it less readable or intuitive, or requires a lot of work.

A compromise may be to allow any form of input and have the program automatically substitute some operators before constructing a tableau. This method requires no extra effort from the user, but does add to the total run time and may lessen readability in a visual representation of a tableau. This leads to wonder what takes more time: substituting operators, or having more rules available when building a tableau?

In order to try to answer this question, a tableau prover will be made for provability logic, with the additional functionality of substituting operators in the input. The prover should run in PSPACE, since GL, like the basic modal logic K, can be shown to be decidable in this complexity (Ladner, 1977). Due to possible difference in complexity of the substitutor part and the tableau builder part of the program, the time difference may vary for formula of different lengths and may depend on the operators substituted and those used in the tableau builder. To test these time differences, experiments will have to include input formulae of different lengths and with different operators, and different program settings will have to be compared. For each input formula and test setting, the time will be recorded with and without substitution.

2 Provability Logic

As stated in the introduction, the provability logic GL is an extension of the basic modal logic K with the Löb axiom $\Box(\Box A \rightarrow A) \rightarrow \Box A$. This has as a consequence that GL is transitive and converse well-founded (Boolos, 1995). The converse well-foundedness of GL means that a frame valid in GL cannot contain an infinite ascending sequence of relations. Due to its transitivity, this also means tableaux in GL cannot have infinite branches.

2.1 GL-tableau rules

For the construction of GL-tableaux, sound and complete rules for GL will be needed. Since GL extends the basic normal logic K, all K-tableau rules still apply. For the basic operators, this means their rules can be used unchanged. Also the transitivity rule is the same as used in Priest (2008). For the modal operators however, the rules will have to change slightly, since they are no longer complete after addition of the GL-axiom to K. The new $\Box$-rules for world-labelled GL-tableaux can be converted from the modal calculus rule for $\Box$, which can be found in literature (Rautenberg, 1983). Since $\Box A = \neg\Diamond\neg A$, the rules for $\Diamond$ can be derived from the rules for $\Box$.

The GL-tableau rules for modal operators are as follows:

$\Box A$, i	$\neg\Diamond A$, i
irj	irj
|	|
$\Box A$, j	$\neg\Diamond A$, j
A, j	$\neg A$, j

For an old j already appearing on the branch, and:

$\neg\Box A$, i	$\Diamond A$, i
|	|
irj	irj
$\Box A$, j	$\neg\Diamond A$, j
$\neg A$, j	A, j

For a new j not yet appearing on the branch.

Note that the rules for $\Box A$ and $\neg\Diamond A$ already include the implications of the transitivity alone, and as such, the transitivity rule is not explicitly needed for a complete tableau.

Basic modal logic can take a while to grasp, but provability logic rules may be even more counter-intuitive. The following may help give some insight in the modal rules for GL (Goré, 1999):

The GL axiom

$$\Box(\Box A \to A) \to \Box A$$

may be rewritten in contraposition

$$\neg\Box A \to \neg\Box(\Box A \to A)$$

which is the same as

$$\Diamond\neg A \to \Diamond(\Box A \wedge \neg A)$$

So if there is a world w where $\Diamond\neg A$ is true, then there must be another world w' accessible from w, where both $\Box A$ and $\neg A$. In other words, if $\neg A$ is true in an accessible world, then there must be a last such accessible world where $\neg A$ is true. For if $\neg A$ was true in a world accessible from this last world, then the $\Box A$ being true in the last world would be contradicted.

2.2 Characterisations of GL

Definition 2.1 (Transitive)**.** For all $F = \langle W, R\rangle$:
$F \models \Box p \to \Box\Box p \Leftrightarrow$ R is transitive.

Definition 2.2 (Converse wellfounded)**.** For all $F = \langle W, R\rangle$:

If for every non-empty $X \subseteq W$ there is an element $x \in X$ such that xRy for no $y \in X$, then R is converse well-founded.

Theorem 2.1. *For all $F = \langle W, R\rangle$:*
R is converse well-founded $\Leftrightarrow$ R contains no infinite sequence of relations $x_0Rx_1Rx_2\ldots$

Proof. $\Longrightarrow$ Suppose R contains an infinite sequence of relations $x_0Rx_1Rx_2\ldots$. Let $X \subseteq W$ be the set of worlds in this infinite sequence. X is then a non-empty set, where for each $x \in X$, there is a $y \in X$ such that xRy. By definition 2.2, R is then not converse well-founded.

$\Longleftarrow$ Suppose R contains no infinite sequence of relations. Suppose $X \subseteq W$ is non-empty and for all $x \in X$, xRy for some $y \in X$. Take any $x_0 \in X$, then there is some $x_1 \in X$ such that x_0Rx_1, resulting in an infinite sequence of relations $x_0Rx_1Rx_2\ldots$. This cannot be, since R contains no infinite sequences, so for all non-empty $X \subseteq W$, there is an $x \in X$ such that xRy for no $y \in X$. Hence R is converse well-founded by definition 2.2. ■

Lemma 2.2. *For all $F = \langle W, R\rangle$:*
$F \models \Box(\Box p \to p) \to \Box p \Rightarrow$ R is transitive.

Proof. Suppose $F \models \Box(\Box p \to p) \to \Box p$, in order to show that R is transitive, we will show that $F \models \Box A \to \Box\Box A$, which is the same by definition 2.1.

Suppose A. If $\Box\Box A \wedge \Box A$, then $\Box A \wedge A$, so $(\Box\Box A \wedge \Box A) \to (\Box A \wedge A)$. This means that $A \to ((\Box\Box A \wedge \Box A) \to (\Box A \wedge A))$ is a propositional tautology.

Since all propositional tautologies are valid in all frames, $F \models A \to ((\Box\Box A \wedge \Box A) \to (\Box A \wedge A))$, and by distribution of $\Box$, $F \models A \to (\Box(\Box A \wedge A) \to (\Box A \wedge A))$. By necessitation, $F \models \Box(A \to (\Box(\Box A \wedge A) \to (\Box A \wedge A)))$, and by distribution of $\Box$, $F \models \Box A \to \Box(\Box(\Box A \wedge A) \to (\Box A \wedge A))$.

Now suppose $F \models \Box(\Box p \to p) \to \Box p$, then by substitution of p with $(\Box A \wedge A)$, $F \models \Box(\Box(\Box A \wedge A) \to (\Box A \wedge A)) \to \Box(\Box A \wedge A)$. Because $F \models \Box A \to \Box(\Box(\Box A \wedge A) \to (\Box A \wedge A))$, by modus ponus $F \models \Box A \to \Box(\Box A \wedge A)$. By distributivity of $\Box$, this implies $F \models \Box A \to \Box\Box A \wedge \Box A$, so $F \models \Box A \to \Box\Box A$. ■

Lemma 2.3. *For all $F = \langle W, R\rangle$:*
$F \models \Box(\Box p \to p) \to \Box p \Rightarrow$ R is converse well-founded.

Proof. Suppose R is not converse well-founded. Then, by theorem 2.1, it contains an infinite sequence of relations $x_0Rx_1Rx_2\ldots$. Let $X \subseteq W$ be the set of worlds in this infinite sequence. Take the evaluation V such that for all $w \in W$, $w \models p$ if and

only if $w \notin X$. Then for any $x \in X$, $x \not\models p$, and consequently $x \not\models \Box p$.

Suppose $x \not\models \Box(\Box p \to p)$, then for some $y \in W$, xRy and $y \not\models \Box p \to p$, and consequently $y \models \Box p$ and $y \not\models p$. Because $y \not\models p$, $y \in X$, and as such $y \not\models \Box p$, causing a contradiction. Hence $x \models \Box(\Box p \to p)$, and since $x \not\models \Box p$, $F \not\models \Box(\Box p \to p) \to \Box p$. ■

Lemma 2.4. *For all $F = \langle W, R\rangle$:*
$F \models \Box(\Box p \to p) \to \Box p \Leftarrow R$ is transitive and converse well-founded.

Proof. Suppose R is transitive and converse well-founded. Suppose $w \not\models \Box p$. Take $X \subseteq W$ as all $x \in W$ where wRx and $x \not\models p$. Because $w \not\models \Box p$, there must be a y such that wRy and $y \not\models p$, so $y \in X$. Then there is a non empty $X \subseteq W$, and because R is converse well-founded, there must be a $x \in X$ such that xRz for no $z \in X$. Since this $x \in X$, wRx and $x \not\models p$.

Suppose xRz, then $z \notin X$. Because R is transitive, this means wRz, and since $z \notin X$, also $z \models p$. Because of this, $x \models \Box p$ is obtained, and since $x \not\models p$, also $x \not\models \Box p \to p$ and $w \not\models \Box(\Box p \to p)$.

In conclusion $w \not\models \Box p \to \Box(\Box p \to p)$, and by contraposition $w \models \Box(\Box p \to p) \to \Box p$. ■

Theorem 2.5. *$\langle W, R\rangle \models \Box(\Box p \to p) \to \Box p \Leftrightarrow R$ is transitive and converse well-founded.*

Proof. By lemma 2.2 and 2.3, *$\langle W, R\rangle \models \Box(\Box p \to p) \to \Box p \Rightarrow R$ is transitive and converse well-founded.*

By lemma 2.4, *$\langle W, R\rangle \models \Box(\Box p \to p) \to \Box p \Leftarrow R$ is transitive and converse well-founded.* ■

2.3 Soundness of GL-tableaux

Definition 2.3. Let $I = \langle W, R, v\rangle$ be any modal interpretation where R is transitive and converse well-founded, and b be any branch of a tableau. Then I is faithful to b iff there is a map, f, from the natural numbers to W such that:

For every node A, i on b, A is true at $f(i)$ in I.
If irj is on b, $f(i)Rf(j)$ in I.

Lemma 2.6 (Soundness Lemma)**.** *Let b be any branch of a tableau, and $I = \langle W, R, v\rangle$ be any interpretation where R is transitive and converse well-founded. If I is faithful to b, and a GL-tableau rule is applied to it, then it produces at least one extension, b', such that I is faithful to b'.*

Proof. Let f be a function which shows I to be faithful to b. For every new tableau rule for GL shall be shown that I is faithful to the resulting extended branch b'.

Suppose $\Box A, i$ is on b, and the new $\Box$-rule for GL is applied. Since I is faithful to b, $\Box A$ is true at $f(i)$. For every sentence on b of the form irj, $f(i)Rf(j)$. If irj is on b for any i and j, the branch is extended with $\Box A, j$ and A, j. Since $\Box A$ is true at $f(i)$, it is also true at $f(j)$ by transitivity, and A is true at $f(j)$ by definition of $\Box$. Hence I is faithful to the extension of b.

Suppose $\Diamond A, i$ appears on b, and the new $\Diamond$-rule for GL is applied to obtain an extended branch b' containing irj; A, j; and $\neg\Diamond A, j$ for some new j. Since I is faithful to b, $\Diamond A$ is true in $f(i)$. Then by definition of $\Diamond A$ and the converse well-foundedness of R, there must be a world w such that $f(i)Rw$, and both A and $\neg\Diamond A$ are true in w. For extend f such that $f(j) = w$ and suppose $\Diamond A$ were true in $f(j)$, then repeated application of $\Diamond$-rule introducing worlds with $\Diamond A$ would result in an infinite branch, which cannot be by definition of converse wellfoundedness. Hence I is faithful to the extension of b.

Suppose $\neg\Box A, i$ appears on branch b, and the new $\neg\Box$-rule for GL is applied to obtain an extended branch b'. Since I is faithful to b, $\neg\Box A$ is true in $f(i)$. Since this is the same as $\Diamond\neg A$ being true in $f(i)$, and the $\Diamond$-rule produces an extension which I is faithful to, I is faithful to the extension of b.

Suppose $\neg\Diamond A, i$ appears on branch b, and the new $\neg\Diamond$-rule for GL is applied to obtain an extended branch b'. Since I is faithful to b, $\neg\Diamond A$ is true in $f(i)$. Since this is the same as $\Box\neg A$ being true in $f(i)$, and the $\Box$-rule produces an extension which I is faithful to, I is faithful to the extension of b.

Suppose both irj and jrk appear on branch b, and the transitivity rule is applied to obtain an extended branch b' containing irk. Since I is faithful to b, $f(i)Rf(j)$ and $f(j)Rf(k)$. Since R is transitive, this means $f(i)Rf(k)$. Hence I is faithful to b'.

■

Theorem 2.7 (Soundness Theorem)**.** *If $GL \vdash A$ then $GL \models A$.*

Proof. Suppose $GL \not\models A$. Then there is an interpretation $I = \langle W, R, v\rangle$ where R is transitive and converse well-founded, that makes A false in some world w. Let f be any function such that $f(0) = w$. This shows I to be faithful to the initial list. By repeated application of the soundness lemma (lemma 2.6), a complete branch can be found such that I is faithful to each initial section of it. If b were closed, then it would have to contain a contradiction in some initial section of b. Since I is faithful to b, this cannot be, so the tableau is open. $GL \nvdash A$. ■

2.4 Completeness of GL-tableaux

Theorem 2.8. *For all $F = \langle W, R\rangle$ where W is finite and R is transitive:*

R is irreflexive $\Leftrightarrow$ R is converse well-founded.

Proof. $\Longrightarrow$ Suppose R is not converse well-founded, by theorem 2.1, R then contains an infinite sequence of relations $x_0Rx_1Rx_2\ldots$. Because W is finite, it follows that for some x_n and x_m in the infinite sequence, $x_n = x_m$. Since then $\ldots Rx_nR\ldots Rx_m\ldots$, by transitivity x_nRx_m, so R is not irreflexive.

$\Longleftarrow$ Suppose R is not irreflexive, then there is some $w \in W$ such that wRw. Then R contains an infinite sequence of relations $wRwRw\ldots$, so R is not converse well-founded by theorem 2.1.

■

Definition 2.4. Let b be an open branch of a tableau, and $I = \langle W, R, v\rangle$ be the interpretation induced by b. $W = \{w_i : i \text{ occurs on } b\}$. w_iRw_j iff irj occurs on b. If p, i occurs on b, then $v_{w_i}(p) = 1$; if $\neg p, i$ occurs on b, then $v_{w_i}(p) = 0$ (and otherwise $v_{w_i}(p)$ can be anything one likes).

Lemma 2.9 (Completeness Lemma)**.** *Let b be any open complete branch of a GL-tableau, and $I = \langle W, R, v\rangle$ be the interpretation induced by b. Then $W \in I$ is finite, and:*

a) if A, i is on b then A is true at w_i

b) if $\neg A, i$ is on b then A is false at w_i

Proof. This will be a proof by induction over the form of A. For each case, it will be shown that the applicable tableau-rule cannot result in an infinite number of worlds.

If A is atomic, then a and b are true by definition. Since there is no applicable rule, this does not result in an infinite number of worlds.

Take two arbitrary sentences P and Q, for which the lemma is true (*induction hypothesis*).

If A is of a propositional form, then a and b are true by the completeness of K (see Priest). Since the propositional tableau-rules only affect a single world, this does not result in an infinite number of worlds.

If A is of the form $\Diamond P$ and $\Diamond P, i$ occurs on b, then the $\Diamond$-rule has been applied to $\Diamond P, i$. This means $\neg\Diamond P, j$; P, j and irj appear on the branch for some new j. Then by the induction hypothesis, P is true at w_j and w_iRw_j. Hence $\Diamond P$ is true at w_i, so a and b are true. Suppose this $\Diamond P$ true at w_i causes introduction of an additional world, then $P = \Diamond Q$, so $\Diamond\Diamond Q$ at w_i. This means, by repeated application, that in order for this sentence to introduce an infinite amount of worlds, the sentence must be infinitely long $\Diamond\Diamond\ldots\Diamond R$, which cannot be, so this sentence does not result in an infinite number of worlds.

If A is of the form $\Box P$ and $\Box P, i$ occurs on b, then for all j such that irj appears on b, $\Box P, j$ and P, j are also on b. So for all w_j such that w_iRw_j, P is true at w_j by the induction hypothesis. Hence $\Box P$ is true at w_i, so a and b are true. Suppose this $\Box P$ true at w_i causes introduction of an additional world, then $P = \Diamond Q$. Then $\Box\Diamond Q, i$, and consequently $\Box\Diamond Q, j$ and $\Diamond Q, j$ appear on the branch. Then consecutive application of the $\Diamond$-rule on $\Diamond Q, j$ and the $\Box$-rule on $\Box\Diamond Q, j$ results in both $\neg\Diamond Q$ and $\Diamond Q$ being true at the new world. This cannot be, so this sentence cannot result in an infinite number of worlds. ■

Theorem 2.10 (Completeness Theorem)**.** *If $GL \models A$ then $GL \vdash A$.*

Proof. Suppose $GL \nvdash A$. Let $I = \langle W, R, V\rangle$ be the interpretation induced by an open branch on a complete tableau. Then, by completeness lemma (lemma 2.9), A is false at w_0, and consequently $GL \not\models A$ if the I is of the required kind, i.e. transitive and converse well-founded.

Suppose w_iRw_j and w_jRw_k for some $w_i, w_j, w_k \in W$. Then irj and jrk appear on the branch, so the transitivity rule has been applied, resulting in irk. Hence w_iRw_k and I is transitive.

Suppose the interpretation is not converse well-founded, then there is some infinite sequence of relations in the interpretation. Then since $W \in I$ is finite by the completeness lemma and I is transitive, I is not irreflexive by 2.8, so wRw for some $w \in W$.

Suppose $w_i R w_j$ for some $w_i, w_j \in W$. Then irj appears on the branch, so the $\Diamond$-rule has been applied. This means $\Diamond A, i$; $\neg\Diamond A, j$; and A, j appear on the branch. Then by completeness lemma, $\Diamond A$ is true in w_i, and $\neg\Diamond A$ and A are true in w_j. Suppose $w_i = w_j$, then both $\Diamond A$ and $\neg\Diamond A$ are true in $w_i = w_j$, leading to a contradiction.

Hence I is converse well-founded.

■

3 Program design

The main focus for the design of the program was to produce tableaux similar to human-made tableaux. That is, for a certain input sentence, the tableau produced by the program should appear like a human could have made it on paper. This means the program would have to apply tableau-rules in a similar order as a human might. When presented with an initial list on a tableau, a human would likely apply simple propositional rules before applying rules that complicate the tableau more (like disjunctions, equivalences or modal rules). Having the program use a similar order, it would often result in tableaux that are shorter than they could have been when for example the rules were applied in order of appearance. Additionally, this could make tableaux easier to read for the user.

Since the program creates a complete tableau every time, as opposed to just checking the validity of the input, every applicable rule must have been applied to open branches on the tableau. This means no rules can be skipped by using fancy tricks in the solver, since this would change the output tableaux, reducing the insight the output can give into why a certain input is valid or not in GL.

Since the output should be intuitive and, as a result, no fancy tricks can be used, the complete program was made to abide by the KISS principle (originally "keep it simple stupid" or later "keep it short and simple"). For the sake of efficiency, the program tries to do as little twice as possible, and traverse the tableau only when it is needed.

3.1 Tableau building algorithm

The input sentence (infix notation) is first parsed into an expression tree by recursive descent. This allows for easy detection of the main operator in each (sub)sentence, since this would be the root node of the tree. The negation of the expression tree is passed to the tableau builder.

This input expression will be in the root of the tableau, which is a binary tree of tableau-nodes. Each tableau-node contains an expression, world, and pointers to the node's children and parent. Whenever a node (this includes the root!) is added to the tableau, it is checked for an applicable rule. If some rule is applicable to the new tableau-node, a task containing pointers to the node and respective rule is pushed onto a priority queue.

The algorithm continuously pops the priority queue, and executes the task (if the input is on an open branch). This will result in new nodes being added to the tableau, and possibly new tasks pushed onto the queue. When the queue is empty, there are no more applicable rules, i.e. the tableau is complete.

When a task is executed, the appropriate tableau rule is applied to its input tableau-node. Each rule creates one or two new (sub-)tableaux, which are copied under each open branch under the input node. If the rule would introduce a new world, then for each branch under the input a new world is introduced. For each newly introduced world, the origin world, from which it was introduced, is saved. After the introduction of a new world and its placement on the tableau, a special rule is called that immediately applies all universal rules ($\Box$ and $\neg\Diamond$) from the origin world to the new world.

Each time a new production is placed on the tableau, the program checks if this causes a contradiction on the branch. This is done by traversing the tableau from leaf to root, checking at each node if it contradicts the new production. If a contradiction is found, the branch is closed. The branch is closed by placing an empty tableau node under the branch, and setting each node on the branch to closed.

In order to minimise the number of duplicate expressions on a branch, no task is pushed onto the queue if the branch already contains another instance of the same expression (i.e. an equivalent task has already been pushed onto the queue).

Duplicates are not completely removed from the tableau, since this could cause branches to disappear completely in some cases. Imagine for instance $A \vee B$ and A appearing on a branch, omission of the duplicate A after application of the $\vee$-rule would have that new branch (where B may be undefined) disappear. The only rules after which duplicate productions are completely omitted are the $\Box$-rule and $\neg\Diamond$-rule, because these two rules are applied separately for each newly introduced world and these rules never split a branch (they just extend it).

If a single rule application would result in two identical branches being produced, only a single branch is added to the tableau. For example, $A \vee A, i$ only adds a single A, i to the tableau, where the $\vee$-rule would usually produce two new extended branches.

See Appendix A for more details and pseudo code of the tableau building algorithm.

3.2 Substitution algorithms

Two kinds of substitution algorithms were implemented in order to test the research question. First of all, an algorithm that converts an expression to negation normal form (NNF), this substitutes all implications ($A \rightarrow B$) and equivalences ($A \leftrightarrow B$) before moving all negations inwards. Secondly, an algorithm was implemented that converts an expression to an equivalent formula using only $\neg$, $\Box$ and $\vee$.

3.2.1 NNF conversion

Conversion to NNF is done in two steps, in each of which the expression tree is traversed exactly once from root to leaves. In the first step, all instances of $A \rightarrow B$ are substituted with $\neg A \vee B$ and all instances of $A \leftrightarrow B$ are substituted with $(\neg A \vee B) \wedge (A \vee \neg B)$.

In the second step, all negations are moved inwards towards the literals. This is done by application of DeMorgan rules, removal of double negations, and substitution of $\neg\Box A$ and $\neg\Diamond A$ with respectively $\Diamond\neg A$ and $\Box\neg A$.

3.2.2 Three operators conversion

In order to obtain the equivalent formula that only contains $\neg$, $\Box$ and $\vee$, all other operators are substituted. This is done in a similar manner as the NNF conversion, in that this algorithm also traverses the expression tree twice. The first step involves substituting all operators in terms of $\neg$, $\Box$ and $\vee$. This is done by substituting $A \rightarrow B$ for $\neg A \vee B$, $A \leftrightarrow B$ for $\neg(\neg(\neg A \vee B) \vee \neg(A \vee \neg B))$, $A \wedge B$ for $\neg(\neg A \vee \neg B)$, and $\Diamond A$ for $\neg\Box\neg A$.

In the second step, all double negations are removed.

4 Experiments

For each input formula, the total CPU time it takes to convert the negated formula to the appropriate form, construct the tableau out of this, and free the tableau and expression tree is measured for each form (unchanged input, NNF, and three operators). Since the tableau is not printed during the experiments (it could be illegibly large, and blow up the output file), the number of nodes and worlds, and whether the tableau is closed is printed instead.

All experiments will be run on Linux Mint 17.3 operating system, using a $4GHz$ CPU and $16GB$ RAM.

4.1 CPU time measurement

CPU time is the time the CPU actually spends on executing a certain program. This is usually different from the elapsed real time, for CPU time does not include time the CPU is waiting for input (or output) or time spent on other processes. Additionally, if a program uses multiple cores, the CPU time of all these cores will be summed.

The measurements of CPU time are done using the clock function from the standard C library ⟨time.h⟩. This function is called before and after the to-be-timed code, and subtraction of the former from the latter results in the number of CPU clock ticks passed. This number of ticks is divided by the constant $CLOCKS\_PER\_SEC$ (from the same library) to obtain the passed CPU time. Due to hardware limitations, the CPU time is not reliable for measuring time under a few milliseconds, while the to-be-timed part of the program in simple cases only takes microseconds. For this reason, the to-be-timed part of the program will actually be executed a thousand times within a single measurement and the average will be reported.

Note that even with this method, the reported CPU time should not be taken as an absolute ($CLOCKS\_PER\_SEC$ is defined as 10^{-6} in many operating systems, regardless of the actual CPU speed). It will however still be in the same order of magnitude as the actual CPU time, and offer insight into the relative runtime of the different algorithms.

4.2 Test formulae

Due to a lack of available benchmark sets for GL, test formulae were hand picked from a variety of sources. First of all, Boolos (1995) offers eight exercises for constructing GL-tableaux. This amount of formulae may seem meagre, but since formulae intended to be proven in GL are sparse, these are included in the experiments. In addition to this, some arbitrary short formulae (mostly propositional and modal axioms) were also tested. Finally, other test formulae come from benchmark sets originally intended for different modal languages, like the Logics Workbench benchmark formulae for K (Balsiger, Heuerding, and Schwendimann, 2000), and a set of benchmarks for PDL (Abate, Goré, and Widmann, 2009).

5 Results

The program can construct a complete GL-tableau for all input formulae in a reasonable time, as long as enough RAM is available on the machine. During testing, the limit for tableau size, i.e. the largest tableau that could be made on the test machine with 16GB available RAM, ranged in the order of 10^7 tableau nodes, depending on formulae lengths. These kind of tableaux result in output files that are far too large to compile to pdf, but they are likely beyond any practical use anyway. See figure 5.1 for an example of a more tractable tableau. To construct this tableau proving the Löb axiom takes approximately $1.9\mu s$ CPU time.

The complexity of the test formulae varied greatly, from single node tableaux, to tableaux that could not be solved due to a lack of RAM. As a matter of fact, most formulae from the LWB benchmark sets for K would run out of memory in either one or more of the three test forms. Fortunately,

$$\neg(\Box(\Box P_0 \to P_0) \to \Box P_0), 0$$
$$\Box(\Box P_0 \to P_0), 0$$
$$\neg\Box P_0, 0$$
$$0r1$$
$$\Box P_0, 1$$
$$\neg P_0, 1$$
$$\Box(\Box P_0 \to P_0), 1$$
$$(\Box P_0 \to P_0), 1$$
$$\neg\Box P_0, 1 \qquad P_0, 1$$
$$\times \qquad \times$$

Figure 5.1: A closed tableau proving the Löb axiom $\Box(\Box P_0 \to P_0) \to \Box P_0$ in GL.

the number of test formulae available was still sufficient to find some interesting results.

For some formulae, the form of the equivalent formula used for tableau construction makes a difference for CPU time and/or tableau size. This can already be seen in simple cases like the tableaux for $\neg(A \leftrightarrow B), i$, where the unchanged formula results in a tableau with only 6 nodes (in $< 1.0\mu s$), but the NNF and 3 operator form result in tableau around twice that size (in respectively $1.9\mu s$ and $2.5\mu s$). This is due to the fact that both the conversion algorithms substitute all equivalences by two disjunctions, leading to four branches on the tableau instead of two.

Another simple case where the form influences tableau size is shown in figure 5.2, here $\Box P, i$ and $\Diamond\neg P, i$ appear on the same branch. While these two contradict each-other, the tableau only closes after another world has been introduced. In the 3 operator form however, $\Diamond\neg P$ is substituted with $\neg\Box P$, which will close the branch instantly when it appears together with $\Box P$.

Although in these last two examples the difference is not very large in an absolute sense, longer sentences can contain multiple instances of these smaller cases, leading to dramatic differences in tableau size between formula forms. For example the tableau to proof $\Box(P \leftrightarrow (\Box(P \vee \Box\bot) \to \Box(P \to \Box\bot))) \to \Box(P \leftrightarrow (\Box\Box\Box\bot \to \Box\Box\bot))$ contains 169 nodes over 9 worlds ($1.2 * 10^2 \mu s$), and the tableau for the 3 operator form is only slightly larger, but the tableau for the NNF explodes into more than 9 million nodes and 400 thousand worlds ($6.8s$).

These differences in run time and tableau size do

$\neg\neg(\Box P_0 \wedge \Diamond\neg P_0), 0$
$(\Box P_0 \wedge \Diamond\neg P_0), 0$
$\Box P_0, 0$
$\Diamond\neg P_0, 0$
$0r1$
$\neg\Diamond\neg P_0, 1$
$\neg P_0, 1$
$\Box P_0, 1$
$P_0, 1$
$\times$

$\neg(\neg\Box P_0 \vee \Box P_0), 0$
$\neg\neg\Box P_0, 0$
$\neg\Box P_0, 0$
$\times$

Figure 5.2: Left: the tableau proving $\neg(\Box P_0 \wedge \Diamond\neg P_0)$ in GL (Nodes: 9, Worlds: 2, Time: $1.5\mu s$). Right: the tableau in 3 operator form (Nodes: 4, Worlds: 1, Time: $< 1.0\mu s$).

not seem to be exclusive to any one or two forms, for any form examples have been found that they can be larger and smaller than either one of the two others (see Appendix B).

6 Conclusion

A program was made that can construct tableaux in GL provability logic. The rules applied by the program were proven to be sound and complete with respect to GL, which in turn was shown to entail all transitive and converse well-founded frames. Three different ways of converting the input before tableau construction were compared in order to test whether conversion to a logically equivalent formula influences run-time. It seems that for some input formulae, the form greatly influences tableau sizes as well as run time.

These differences in run time can arise when for example the input sentence contains equivalences, since each substituted equivalence can cause the tableau to split into more branches. Another thing that can influence the difference in run-time between forms is the amount of $\Box$- and $\Diamond$-operators in a formula, as having the two types of operators appear together in a formula can make it more difficult for the program to find contradictions and close branches.

These (and likely also other) small influences can add up quickly in longer and more complex formulae, causing the tableau for some inputs and forms to explode. However, the possibility for the tableau to explode does not seem exclusive to any of the three forms, as no one of the three forms is guaranteed to be the least complex. Where the unchanged input is the only form that can contain equivalences, the 3 operator form is the only form guaranteed to not contain any $\Diamond$-operators. The NNF can either contain less or more $\Diamond$-operators than $\Box$-operators, as these two are substituted by eachother when a $\neg$-operator is moved inwards over them. It seems that the presence (or absence) of a single $\neg$-operator somewhere in a formula can greatly influence the tableau size and run time this way.

Additionally, the lack of a conventional benchmark method for GL-theorem provers makes it hard to generalise results over certain classes of formulae, since these classes were not defined for GL. The formulae from other existing benchmark sets are divided in different classes, but it is in no way guaranteed that the GL-tableaux will turn out similar to the tableaux for the intended language of the set. Likewise, for most of the test sentences, their satisfiability in GL is not known before testing. For these reasons, the results cannot be generalised over different formulae, as this will unlikely be very meaningful. This also means that the complexity of the algorithm cannot be demonstrated empirically.

However, even though the results cannot be generalised over formulae, and as such the tableau length can also not be predicted based on form, the results do illustrate the importance of how an input formula is formulated before constructing a tableau in GL.

7 Discussion

The performance of the program is hard to compare to other GL-theorem provers, not only due to the lack of benchmark sets, but also due to the fact that most theorem provers do not construct and report complete tableaux. The program can solve problems quickly, but it can also run out of memory within mere seconds, because it has to store a complete tableau (this is also why many of the Logics Workbench (LWB) benchmark problems for K were too complex to solve in some forms). Although a lot of theorem provers (for GL or other modal languages), for example the one by Goré and Kelly (2007), are inspired by tableau methods and use

similar rules, these typically do not store the complete tableaux, enabling them to solve more complex problems (albeit in much more time) before running out of memory. During experiments with their tableau rules, Goré and Kelly also noted that for some small rule changes, the run time would decrease for some input formulae, where for other inputs it would increase severely. Interestingly, when they assumed all formulae were first put into NNF (as they note is traditional in automated theorem proving) they also ran into non-terminating problems. Since their research is from more than 10 years ago, using a framework that is even older (the Tableau Workbench (Abate and Goré, 2003)) and quite high level, their non-terminating issues might have been related to efficiency problems (the proof just being very very large, instead of actually non-terminating), or simply Moore's law, by which 10 years would mean approximately $2^5 = 32$ times faster hardware.

The traditional conversion of the input to NNF is likely a bad idea for use with GL-theorem provers, as this can make the search space explode when used in conjunction with certain rules. However, based solely on these results, there is no reason to assume this applies to any other (modal) language as well (besides the (likely) almost inevitable increase of size when substituting an equivalence).

In future research, it may be interesting to compare the performance of the program with different sets of (sound and complete) rules. For example, what would be different if the rules used for the negations of the modal operators had used less shortcuts? For example if the $\neg\Box$-rule would have simply produced an expression of the form $\Diamond\neg A$, as opposed to introducing new worlds itself.

The input form has been shown to have influence on tableau complexity, but the tested forms could not be said to be consistently faster or slower. This leads to wonder if there are any forms that do guarantee the tableau cannot explode relative to the unchanged input. For example, allowing equivalences in the three (then four) operator form, or substituting all instances of $\Diamond$ after conversion to NNF.

Finally, results of comparing these different ways to construct GL-tableau would likely be more meaningful and even more interesting, if there was a conventional benchmark set for GL-theorem provers. A benchmark set similar to the LWB benchmark sets for K would be ideal. This benchmark method's sets are procedurally generated within the LWB, using specific branching formulae around core formulae that are known to be either satisfiable or unsatisfiable. Since the LWB does support working with GL, and the code for the generation of formulae is available for K, KT and S4, it should be possible to generate a similar benchmark set for GL in the LWB.

References

Pietro Abate and Rajeev Goré. The tableaux work bench. *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 230–236, 2003.

Pietro Abate, Rajeev Goré, and Florian Widmann. An on-the-fly tableau-based decision procedure for PDL-satisfiability. *Electronic Notes in Theoretical Computer Science*, 231:191–209, 2009.

Peter Balsiger, Alain Heuerding, and Stefan Schwendimann. A benchmark method for the propositional modal logics K, KT, S4. *Journal of Automated Reasoning*, 24(3):297–317, 2000.

George Boolos. *The logic of provability*. Cambridge university press, 1995.

Rajeev Goré. Tableau methods for modal and temporal logics. In *Handbook of Tableau Methods*, pages 297–396. Springer, 1999.

Rajeev Goré and Jack Kelly. Automated proof search in Gödel-Löb provability logic. In *British Logic Colloquium*, 2007.

Richard E Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM Journal on Computing*, 6(3):467–480, 1977.

Graham Priest. *An Introduction to Non-Classical Logic: From If to Is*. Cambridge University Press, 2008.

Wolfgang Rautenberg. Modal tableau calculi and interpolation. *Journal of Philosophical Logic*, 12 (4):403–423, 1983.

Robert M Solovay. Provability interpretations of modal logic. *Israel Journal of Mathematics*, 25 (3):287–304, 1976.

Rineke (L.C.) Verbrugge. Provability logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, summer 2017 edition, 2017.

Appendix A

1 Input parser:

1.1 Default input format

operator	input
$\neg$	~
$\wedge$	&
$\vee$	|
$\rightarrow$	>
$\leftrightarrow$	=
$\Box$	[]
$\Diamond$	<>
$\perp$	#

All atoms must be capital letters, optinally followed by an integer. Input is terminated by whitespace.

1.2 Parser

A recursive descent parser for an expression tree (ExpTree in the pseudocode) was implemented according to the following grammar:

$$Expression ::= [Exp2|(Expression \leftrightarrow Exp2)]$$

$$Exp2 ::= [Exp3|(Exp2' \rightarrow' Exp3)]$$

$$Exp3 ::= [Exp4|(Exp3' \vee' Exp4)]$$

$$Exp4 ::= [Exp5|(Exp4' \wedge' Exp5)]$$

$$Exp5 ::= [Exp6|'\neg' Exp5|'\Box' Exp5|'\Diamond' Exp5]$$

$$Exp6 ::= [Atom|'\perp'|'('Expression')']$$

2 Tableau Builder

2.1 Abstract Data Types

2.1.1 Tableau

TableauNode

ExpTree expression;
Tableau left, right, parent; ▷ **Tableau** is a pointer to a **TableauNode**
boolean closed;
int world;

2.1.2 Task

Task

Tableau input;	
void rulePtr;	▷ a pointer to an applyRule function
int priority;	▷ priorities for the rules described at Heap

2.1.3 Heap

Priority of operations:

- $\neg\neg$
- $\wedge$
- $\neg\vee$
- $\neg \rightarrow$
- $\vee$
- $\neg\wedge$
- $\rightarrow$
- $\leftrightarrow$
- $\neg \leftrightarrow$
- $\Diamond$ and $\neg\Box$

2.1.4 Worlds

Worlds

int nWorlds;
int array introducedBy; ▷ stores for each world by which other world it was introduced

2.2 Functions

```
▷ Returns a complete tableau for the input. Worlds is a reference parameter
and will remember where each world was introduced from.
procedure buildTableau(ExpTree input, Worlds worlds))
    introduceNewWorld(worlds, -1);
    tableau ← newTableau(input, new world);
    heap ← newHeap;
    task ← newTask(tableau);
    if task has an applicable rule then
        pushHeap(tableau);
    end if
    while heap is not empty do
        task ← popHeap;
        if task.input is not on a closed branch then
            task.rule(task.input, heap, worlds);
        end if
    end while
    return tableau;
end procedure
```

```
                    ▷ Returns new Task, which can be added to the heap. Will
make task.rulePtr point to a rule function, iff input is NOT a literal, absurd
or universal statement.
procedure newTask(Tableau input)
    task.input ← newTableau(input, new world);
    make task.rulePtr point to the appropriate rule for input.expression (if
any)
    if task has a rulePtr then
        set task.priority to the priority of task.rulePtr
    end if
    return task;
end procedure
```

```
                                          ▷ Basic form of all non-modal rules.
procedure applyBasicRule(Tableau tableau, Heap heap)
    newProductions ← newTableau(newExpression, tableau.world);
                                    ▷ newProduction can be 1 or 2 variables
    addUnderTableau(newProductions, tableau, heap)
end procedure
```

```
                                ▷ Basic form of a universal rule (i.e. □ or ¬◇.
procedure applyUniversalRule(Tableau in, Tableau under, Heap
heap)
    newProductions ← newTableau(newExpression, under.world);
    addUnderTableau(newProductions, under, heap)
end procedure
```

```
                                    ▷ Basic form of a world introducing rule (i.e. ◇ or ¬□.
procedure APPLYWORLDINTRODUCINGRULE(Tableau tableau, Heap heap, Worlds worlds)
    newProductions ← newTableau(newExpression, tableau.world);
                                    ▷ storing the origin world in newProductions
    ADDWORLDUNDERTABLEAU(newProductions, tableau, heap, worlds)
end procedure
```

```
    ▷ Applies all applicable universal rules from originWorld to the new world.
procedure APPLYALLUNIVERSALRULES(Tableau in, Heap heap, int originWorld)
    tableau ← in.parent;
    while tableau.world does not equal originWorld do
        tableau ← tableau.parent;
    end while
    while tableau.world equals originWorld do
        if tableau contains a universal statement then
            APPLYUNIVERSALRULE(tableau, in, heap);
        end if
        tableau ← tableau.parent;
    end while
end procedure
```

```
        ▷ Adds the specified new productions in all branches under a specified
tableau node. If a rule can be applied to a new tableau node, it is added to
the heap.
procedure ADDUNDERTABLEAU(Tableau new, Tableau under, Heap heap)
    if under has no children then
        put a copy of new as child(s) of under
        if this did not cause a contradiction then
            task ← newTask(tableau);
            if task has an applicable rule then
                pushHeap(tableau);
            end if
        else
            close the branch
        end if
        return under;
    end if
    under.left ← ADDUNDERTABLEAU(new, under.left, heap);
    under.right ← ADDUNDERTABLEAU(new, under.right, heap);
    return under;
end procedure
```

▷ Introduces a new world with the specified new productions in all branches under a specified tableau node. If a rule can be applied to a new tableau node, it is added to the heap. All applicable universal rules are applied afterwards.

```
procedure addWorldUnderTableau(Tableau new, Tableau under, Heap heap, Worlds worlds)
    if under has no children then
        under.left ← copy(new);
        under.left.world ← = introduceNewWorld(worlds, new.world);
        if this did not cause a contradiction then
            task ← newTask(tableau);
            if task has an applicable rule then
                pushHeap(tableau);
            end if
            applyAllUniversalRules(under.left, heap, new.world)
        else
            close the branch
        end if
        return under;
    end if
    under.left ← addWorldUnderTableau(new, under.left, heap, worlds);
    under.right ← addWorldUnderTableau(new, under.right, heap, worlds);
    return under;
end procedure
```

Appendix B

- Clean input: $\neg\neg(A_0 \leftrightarrow B_0)$
- Nodes: 6, Worlds: 1, Closed: no
- 3OP input: $\neg(\neg(A_0 \vee \neg B_0) \vee \neg(\neg A_0 \vee B_0))$
- Nodes: 13, Worlds: 1, Closed: no
- NNF input: $((A_0 \vee \neg B_0) \wedge (\neg A_0 \vee B_0))$
- Nodes: 11, Worlds: 1, Closed: no

- Clean input: $\neg\neg(\Box P_0 \wedge \Diamond\neg P_0)$
- Nodes: 9, Worlds: 2, Closed: yes
- 3OP input: $\neg(\neg\Box P_0 \vee \Box P_0)$
- Nodes: 4, Worlds: 1, Closed: yes
- NNF input: $(\Box P_0 \wedge \Diamond\neg P_0)$
- Nodes: 8, Worlds: 2, Closed: yes

- Clean input: $\neg\neg(\Box P_0 \leftrightarrow \Diamond\neg P_0)$
- Nodes: 16, Worlds: 3, Closed: yes
- 3OP input: $\neg(\neg(\Box P_0 \vee \Box P_0) \vee \neg(\neg\Box P_0 \vee \neg\Box P_0))$
- Nodes: 8, Worlds: 1, Closed: yes
- NNF input: $((\Box P_0 \vee \Box P_0) \wedge (\Diamond\neg P_0 \vee \Diamond\neg P_0))$
- Nodes: 10, Worlds: 2, Closed: yes

- Clean input: $\neg(\Box P_0 \leftrightarrow \Diamond\neg P_0)$
- Nodes: 9, Worlds: 3, Closed: no
- 3OP input: $(\neg(\Box P_0 \vee \Box P_0) \vee \neg(\neg\Box P_0 \vee \neg\Box P_0))$
- Nodes: 10, Worlds: 2, Closed: no
- NNF input: $((\Diamond\neg P_0 \wedge \Diamond\neg P_0) \vee (\Box P_0 \wedge \Box P_0))$

- Nodes: 9, Worlds: 2, Closed: no

- Clean input: $\neg\Box(P_0 \leftrightarrow (\Box\Box\Box\bot \rightarrow \Box\Box\bot))$
- Nodes: 87, Worlds: 7, Closed: no
- 3OP input: $\neg\Box\neg(\neg(P_0\vee\neg(\neg\Box\Box\Box\bot\vee\Box\Box\bot))\vee\neg(\neg P_0\vee(\neg\Box\Box\Box\bot\vee\Box\Box\bot)))$
- Nodes: 122, Worlds: 7, Closed: no
- NNF input: $\Diamond((\neg P_0 \wedge (\Diamond\Diamond\Diamond\neg\bot \vee \Box\Box\bot)) \vee (P_0 \wedge (\Box\Box\Box\bot \wedge \Diamond\Diamond\neg\bot)))$
- Nodes: 4833, Worlds: 324, Closed: no

- Clean input: $\neg(\Box(P_0 \leftrightarrow (\Box(P_0 \vee \Box\bot) \rightarrow \Box(P_0 \rightarrow \Box\bot))) \rightarrow \Box(P_0 \leftrightarrow (\Box\Box\Box\bot \rightarrow \Box\Box\bot)))$
- Nodes: 169, Worlds: 9, Closed: yes
- 3OP input: $\neg(\neg\Box\neg(\neg(P_0 \vee \neg(\neg\Box(P_0 \vee \Box\bot) \vee \Box(\neg P_0 \vee \Box\bot))) \vee \neg(\neg P_0 \vee (\neg\Box(P_0\vee\Box\bot)\vee\Box(\neg P_0\vee\Box\bot))))\vee\Box\neg(\neg(P_0\vee\neg(\neg\Box\Box\Box\bot\vee\Box\Box\bot))\vee\neg(\neg P_0\vee(\neg\Box\Box\Box\bot \vee \Box\Box\bot))))$
- Nodes: 243, Worlds: 9, Closed: yes
- NNF input: $(\Box((P_0\vee(\Box(P_0\vee\Box\bot)\wedge\Diamond(P_0\wedge\Diamond\neg\bot)))\wedge(\neg P_0\vee(\Diamond(\neg P_0\wedge\Diamond\neg\bot)\vee\Box(\neg P_0\vee\Box\bot))))\wedge\Diamond((\neg P_0\wedge(\Diamond\Diamond\Diamond\neg\bot\vee\Box\Box\bot))\vee(P_0\wedge(\Box\Box\Box\bot\wedge\Diamond\Diamond\neg\bot))))$
- Nodes: 9013307, Worlds: 416098, Closed: yes

- Clean input: $\neg((((((\Box(\Box P_0 \vee \Box\Diamond\neg P_0) \vee \Diamond\Box\bot) \vee \Diamond(\Box P_0 \wedge \Diamond\Diamond\neg P_0)) \vee \Diamond(\Box\Diamond P_0 \wedge \Diamond\Diamond\Box\neg P_0)) \vee \Diamond(\Box P_0 \wedge \Box\neg P_0)) \vee \Diamond(\Box(\Box\neg P_0 \vee P_0) \wedge \Diamond\Diamond(\Diamond P_0 \wedge \neg P_0))) \vee \Diamond(\Box(\Diamond\neg P_0 \vee P_0) \wedge \Diamond\Diamond(\Box P_0 \wedge \neg P_0)))$
- Nodes: 1628735, Worlds: 70002, Closed: yes
- 3OP input: $\neg((((((\Box(\Box P_0 \vee \Box\neg\Box P_0) \vee \neg\Box\neg\Box\bot) \vee \neg\Box(\neg\Box P_0 \vee \Box\Box P_0)) \vee \neg\Box(\neg\Box\neg\Box\neg P_0 \vee \Box\Box\neg\Box\neg P_0)) \vee \neg\Box(\neg\Box P_0 \vee \neg\Box\neg P_0)) \vee \neg\Box(\neg\Box(\Box\neg P_0 \vee P_0) \vee \Box\Box(\Box\neg P_0 \vee P_0))) \vee \neg\Box(\neg\Box(\neg\Box P_0 \vee P_0) \vee \Box\Box(\neg\Box P_0 \vee P_0)))$
- Nodes: 5426, Worlds: 248, Closed: yes
- NNF input: $((((((\Diamond(\Diamond\neg P_0\wedge\Diamond\Box P_0)\wedge\Box\Diamond\neg\bot)\wedge\Box(\Diamond\neg P_0\vee\Box\Box P_0))\wedge\Box(\Diamond\Box\neg P_0\vee\Box\Box\Diamond P_0)) \wedge \Box(\Diamond\neg P_0 \vee \Diamond P_0)) \wedge \Box(\Diamond(\Diamond P_0 \wedge \neg P_0) \vee \Box\Box(\Box\neg P_0 \vee P_0))) \wedge \Box(\Diamond(\Box P_0 \wedge \neg P_0) \vee \Box\Box(\Diamond\neg P_0 \vee P_0)))$
- Nodes: 405645, Worlds: 17238, Closed: yes

- Clean input: $\neg(((((\Box P_1 \vee \Box P_2) \vee \Box P_3) \vee \Box P_5) \vee ((((((((((((\Diamond(\neg P_1 \wedge \Box P_1) \vee \Diamond(\neg P_1 \wedge \Box P_3)) \vee \bot) \vee (\bot \vee \bot)) \vee ((\Diamond(\neg P_3 \wedge \Box P_1) \vee \Diamond(\neg P_3 \wedge \Box P_3)) \vee \bot)) \vee (\bot \vee \bot)) \vee ((\Diamond(\neg P_5 \wedge \Box P_1) \vee \Diamond(\neg P_5 \wedge \Box P_3)) \vee \bot)) \vee (\bot \vee \bot)) \vee (\bot \vee \bot)) \vee (\bot \vee (\Diamond(\neg P_4 \wedge \Box P_2) \vee \Diamond(\neg P_6 \wedge \Box P_2)))) \vee (\bot \vee \bot)) \vee (\bot \vee (\Diamond(\neg P_4 \wedge \Box P_4) \vee \Diamond(\neg P_6 \wedge \Box P_4)))) \vee (\bot \vee \bot)) \vee (\bot \vee (\Diamond(\neg P_4 \wedge \Box P_6) \vee \Diamond(\neg P_6 \wedge \Box P_6))))) \vee (((\Diamond\Diamond\neg P_2 \vee \Diamond\Diamond\neg P_4) \vee \Diamond\Diamond\neg P_5) \vee \Diamond\Diamond\neg P_6))$
- Nodes: 155, Worlds: 2, Closed: yes
- 3OP input: $\neg(((((\Box P_1 \vee \Box P_2) \vee \Box P_3) \vee \Box P_5) \vee ((((((((((((\neg\Box(P_1 \vee \neg\Box P_1) \vee \neg\Box(P_1 \vee \neg\Box P_3)) \vee \bot) \vee (\bot \vee \bot)) \vee ((\neg\Box(P_3 \vee \neg\Box P_1) \vee \neg\Box(P_3 \vee \neg\Box P_3)) \vee \bot)) \vee (\bot \vee \bot)) \vee ((\neg\Box(P_5 \vee \neg\Box P_1) \vee \neg\Box(P_5 \vee \neg\Box P_3)) \vee \bot)) \vee (\bot \vee \bot)) \vee (\bot \vee \bot)) \vee (\bot \vee (\neg\Box(P_4 \vee \neg\Box P_2) \vee \neg\Box(P_6 \vee \neg\Box P_2)))) \vee (\bot \vee \bot)) \vee (\bot \vee (\neg\Box(P_4 \vee \neg\Box P_4) \vee \neg\Box(P_6 \vee \neg\Box P_4)))) \vee (\bot \vee \bot)) \vee (\bot \vee (\neg\Box(P_4 \vee \neg\Box P_6) \vee \neg\Box(P_6 \vee \neg\Box P_6))))) \vee (((\neg\Box\Box P_2 \vee \neg\Box\Box P_4) \vee \neg\Box\Box P_5) \vee \neg\Box\Box P_6))$
- Nodes: 1716, Worlds: 18, Closed: yes
- NNF input: $(((((\Diamond\neg P_1 \wedge \Diamond\neg P_2) \wedge \Diamond\neg P_3) \wedge \Diamond\neg P_5) \wedge ((((((((((((\Box(P_1 \vee \Diamond\neg P_1) \wedge \Box(P_1 \vee \Diamond\neg P_3)) \wedge \neg\bot) \wedge (\neg\bot \wedge \neg\bot)) \wedge ((\Box(P_3 \vee \Diamond\neg P_1) \wedge \Box(P_3 \vee \Diamond\neg P_3)) \wedge \neg\bot)) \wedge (\neg\bot \wedge \neg\bot)) \wedge ((\Box(P_5 \vee \Diamond\neg P_1) \wedge \Box(P_5 \vee \Diamond\neg P_3)) \wedge \neg\bot)) \wedge (\neg\bot \wedge \neg\bot)) \wedge (\neg\bot \wedge \neg\bot)) \wedge (\neg\bot \wedge (\Box(P_4 \vee \Diamond\neg P_2) \wedge \Box(P_6 \vee \Diamond\neg P_2)))) \wedge (\neg\bot \wedge \neg\bot)) \wedge (\neg\bot \wedge (\Box(P_4 \vee \Diamond\neg P_4) \wedge \Box(P_6 \vee \Diamond\neg P_4)))) \wedge (\neg\bot \wedge \neg\bot)) \wedge (\neg\bot \wedge (\Box(P_4 \vee \Diamond\neg P_6) \wedge \Box(P_6 \vee \Diamond\neg P_6))))) \wedge (((\Box\Box P_2 \wedge \Box\Box P_4) \wedge \Box\Box P_5) \wedge \Box\Box P_6))$
- Nodes: 112, Worlds: 2, Closed: yes